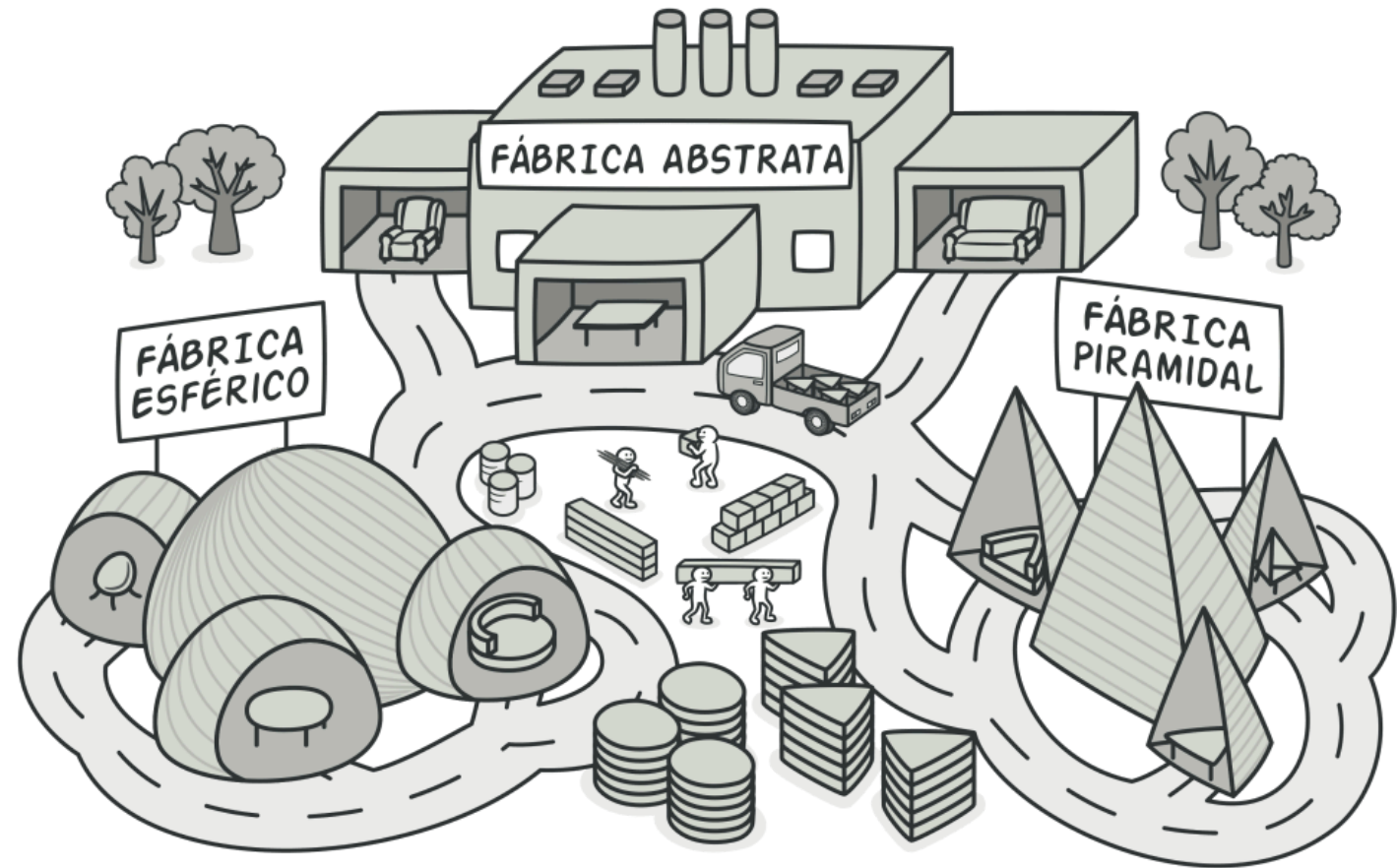


TÓPICO 07: ABSTRACT FACTORY

Design Patterns - Professor Ramon Venson - SATC 2026.2

Definição

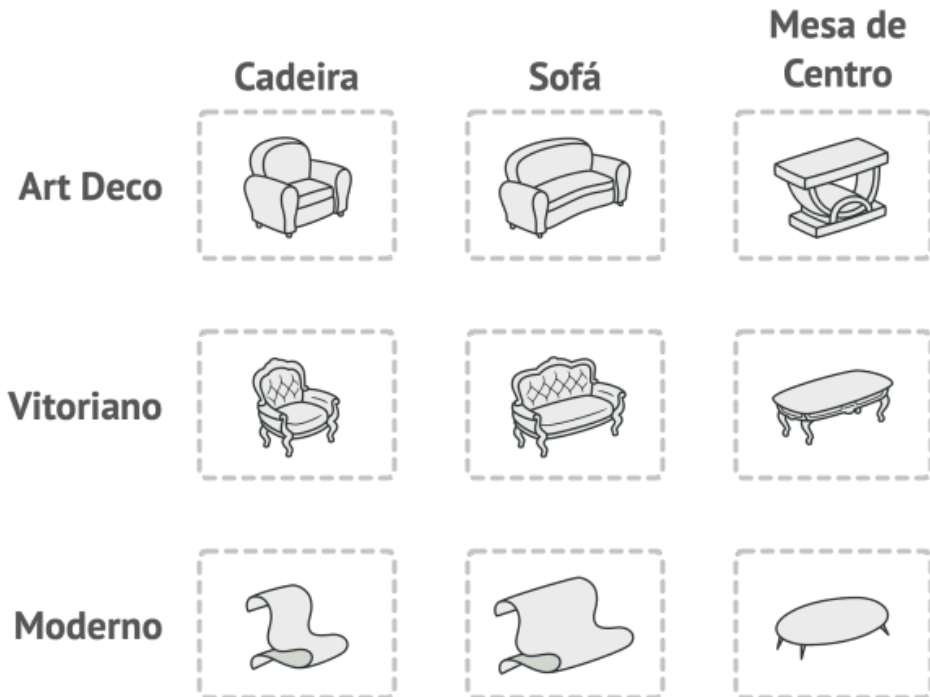
Abstract Factory é um padrão de projeto **criacional** que fornece uma **interface** para **criar famílias de objetos relacionados** sem especificar suas **classes concretas**.



Problema

Alguns sistemas precisam de **famílias de objetos que combinem entre si** (ex.: uma sala com cadeira, sofá e mesa do **mesmo estilo**). Criar cada peça com `new` diretamente leva a:

- **mistura indevida de estilos** (cadeira moderna + sofá vitoriano)
- alto **acoplamento** com as classes concretas
- revisar **vários pontos** do código a cada nova família



Neste exemplo, os móveis da sala nascem de `new` espalhados, e nada garante que o **estilo seja coerente**:

```
public class Sala {  
    private Cadeira cadeira = new CadeiraModerna();  
    private Sofa sofa = new SofaVitoriano();    // estilos misturados!  
    private MesaDeCentro mesa = new MesaDeCentroArtDeco();  
  
    public void exibir() { /* ... */ }  
}
```

- trocar o estilo da sala exige alterar **cada campo**
- nada impede **misturar famílias** por engano

Solução

Vamos **isolar a criação por família**: uma **fábrica abstrata** declara métodos de criação para **todos os produtos** da família, e cada **fábrica concreta** cria apenas produtos **compatíveis entre si**.

- **Produtos abstratos**: interfaces dos produtos (Cadeira, Sofá, Mesa)
- **Fábrica abstrata**: métodos `criarCadeira()`, `criarSofa()`, ...
- **Fábrica concreta**: uma por família/estilo
- **Cliente**: usa apenas abstrações

Ao escolher uma fábrica, o cliente recebe **sempre a mesma família**:

- `FabricaMobiliaModerna` → cadeira + sofá + mesa **modernos**
- `FabricaMobiliaVitoriana` → cadeira + sofá + mesa **vitorianos**
- o cliente **não sabe** nem se importa com a classe concreta
- trocar a família = trocar **uma** fábrica



Implementação

1. Vamos definir as **interfaces de produto** da família:

```
public interface Cadeira {  
    void assentar();  
}  
  
public interface Sofa {  
    void deitar();  
}  
  
public interface MesaDeCentro {  
    void apoiar();  
}
```

2. Criamos os **produtos concretos** de cada família:

```
public class CadeiraModerna implements Cadeira {  
    @Override  
    public void assentar() {  
        System.out.println("Cadeira moderna");  
    }  
}  
  
public class CadeiraVitoriana implements Cadeira {  
    @Override  
    public void assentar() {  
        System.out.println("Cadeira vitoriana");  
    }  
}
```

3. Agora a **fábrica abstrata**, com um método de criação para cada produto:

```
public interface FabricaMobilia {  
    Cadeira criarCadeira();  
    Sofa criarSofa();  
    MesaDeCentro criarMesaDeCentro();  
}
```

As assinaturas retornam **produtos abstratos**, nunca classes concretas.

4. As **fábricas concretas** implementam a interface e produzem apenas a sua família:

```
public class FabricaMobiliaModerna implements FabricaMobilia {  
    @Override  
    public Cadeira criarCadeira() { return new CadeiraModerna(); }  
    @Override  
    public Sofa criarSofa() { return new SofaModerno(); }  
    @Override  
    public MesaDeCentro criarMesaDeCentro() {  
        return new MesaDeCentroModerna();  
    }  
}
```

Utilização

```
FabricaMobilia fabrica = new FabricaMobiliaModerna();  
fabrica.criarCadeira().assentar();  
fabrica.criarSofa().deitar();  
fabrica.criarMesaDeCentro().apoiar();
```

Dessa forma, o cliente **não conhece** as classes concretas, e a **coerência da família** é garantida. Sempre que quiser trocar o estilo, basta trocar a fábrica abstrata.

Configurador

Uma classe cliente pode receber a **fábrica** no construtor e criar todos os produtos de uma vez, ao qual chamamos de **configurador**:

```
public class ConfiguradorDeSala {  
    private final Cadeira cadeira;  
    private final Sofa sofa;  
    private final MesaDeCentro mesa;  
  
    public ConfiguradorDeSala(FabricaMobilia fabrica) {  
        this.cadeira = fabrica.criarCadeira();  
        this.sofa = fabrica.criarSofa();  
        this.mesa = fabrica.criarMesaDeCentro();  
    }  
  
    public void exibir() {  
        cadeira.assentar(); sofa.deitar(); mesa.apoiar();  
    }  
}
```

A escolha da fábrica concreta é feita **uma vez** na inicialização do cliente:

```
FabricaMobilia fabrica = new FabricaMobiliaModerna();  
ConfiguradorDeSala configurador = new ConfiguradorDeSala(fabrica);  
configurador.exibir();
```

Todos os produtos criados pelo configurador serão **coerentes** entre si e de uma única vez.

Mapa de Configuração

A fábrica concreta pode ser escolhida por **configuração** (arquivo, propriedade de sistema, `ServiceLoader`):

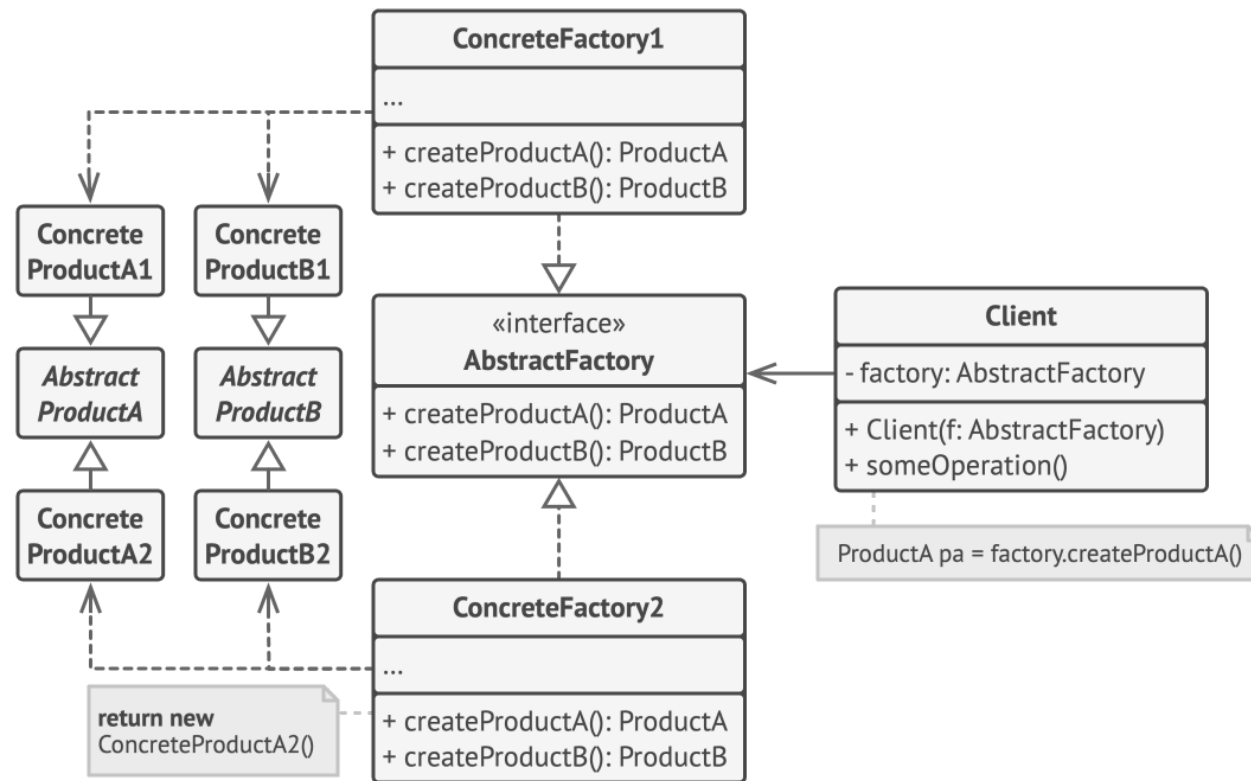
```
String estilo = config.get("mobilia.estilo"); // "moderno"

Map<String, FabricaMobilia> fabricas = Map.of(
    "moderno",    new FabricaMobiliaModerna(),
    "vitoriano",  new FabricaMobiliaVitoriana()
);

FabricaMobilia fabrica = fabricas.getOrDefault(estilo, new FabricaMobiliaModerna());
```

- permite **trocar a família sem recompilar**
- muito usado em temas, drivers e provedores

Diagramas



- **AbstractFactory**: métodos de criação de cada produto
- **ConcreteFactory**: fábrica de uma família específica
- **AbstractProduct**: interfaces dos produtos
- **ConcreteProduct**: implementações por família
- **Client**: usa apenas abstrações (fábrica + produtos)

Analogias

Coleção de roupas:

- uma coleção de verão tem peças que **combinam entre si**
- misturar verão + inverno no mesmo look "quebra" a harmonia
- a fábrica é a **coleção**; as peças são os produtos
- Assim como não se misturam estilos de coleção, **não se misturam famílias** de produtos

Aplicações

- Famílias de objetos que precisam ser **compatíveis entre si**
- Trocar **tema/ambiente/fornecedor** sem reescrever o cliente
- Código que não deve **depende de classes concretas**

Exemplos típicos:

- GUI multi-plataforma (Windows/macOS/Linux)
- conectores de banco (MySQL/PostgreSQL): conexão, comando, transação
- integrações com provedores (AWS/Azure/GCP) com seus serviços

Prós

- **Coerência garantida:** produtos da mesma fábrica sempre combinam
- **Menos acoplamento:** cliente trabalha com abstrações
- **Fácil trocar famílias:** mudança localizada (inicialização)
- Ajuda a aplicar **SRP** e **OCP**

Contras

- **Muitas classes e interfaces** novas junto com o padrão
- Adicionar um **novo produto** na família exige alterar a fábrica abstrata e **todas** as fábricas concretas
- Pode ser **overengineering** quando não há famílias reais

Regra: **vale a pena** quando existem famílias que precisam combinar.

Resumo

- Abstract Factory: **cria famílias de objetos relacionados**
- Fábricas concretas garantem **coerência entre produtos**
- Cliente usa **apenas abstrações**, troca de família é localizada
- Diferente do Factory Method, que cria **um produto** por vez